ИТМО ВТ

Графический интерфейс

AWT

- AWT — Abstract Window Toolkit
  - библиотека, зависимая от графической подсистемы ОС
  - должна одинаково «хорошо» выглядеть на разных платформах

2

В Java применяются разные графические библиотеки, рассмотрим четыре из них. AWT (Abstract Window Toolkit) — первая библиотека для работы с графикой, появилась еще в версии 1.0, и оказалась не совсем удачной. Хоть она и была кроссплатформенной, AWT зависела от графической подсистемы. Предполагалось, что компоненты будут выглядеть одинаково хорошо на разных платформах. На самом деле получилось, что все выглядит одинаково плохо и хуже того, по-разному. Разработчики оставили только те компоненты и только те функции, которые поддерживались во всех операционных системах. В итоге получилась библиотека с очень урезанными возможностями. Компонентов было мало, при этом они поддерживали не все возможные функции. Но зато эта библиотека работала для того времени довольно быстро.

Swing

- Swing
  - надстройка над AWT в виде легковесных Java-компонентов
  - изменяемый вид компонентов

3

С версии 1.1 появилась библиотека Swing, сначала как отдельная библиотека, потом как часть Java. Основная особенность Swing в том, что почти все компоненты написаны на Java. Поэтому в принципе они должны выглядеть одинаково везде. Но, из-за того, что компоненты нужно отрисовывать, все стало работать медленнее, чем было в AWT. Но потом провели работу над ускорением и сейчас правильно настроенный Swing работает достаточно быстро. Полезной особенностью Swing является изменяемый вид компонентов, который можно менять на ходу. Не сказать, что Swing является самой удачной библиотекой, но изучить на его примере основные принципы вполне можно.

SWT

- SWT
  - Часть Eclipse, компоненты-оболочки для компонентов ОС
  - Недостающий функционал написан на Java

4

Библиотека SWT появилась как часть Eclipse, ее разработку поддержала компания IBM. SWT есть не для всех платформ, но при ее создании разработчики удачно соединили лучшее из AWT и Swing. В SWT компоненты и их функции, которые поддерживаются графической подсистемой, как и в AWT работают через адаптеры, а недостающие функции, как в Swing, дописаны на Java.

JavaFX

- JavaFX
  - новая графическая библиотека
  - улучшенная поддержка анимации
  - визуальные эффекты
  - XML для задания интерфейса
  - CSS для задания стилей

5

JavaFX изначально позиционировалась как новая графическая библиотека, с поддержкой анимации, визуальных эффектов, возможностью задания интерфейса с помощью XML, поддержки стилей. Сначала JavaFX была отдельной библиотекой, потом стала частью Java, сейчас снова выделена в отдельной проект под названием OpenJFX.

Создание графических приложений

1) Создание основного окна
2) Создание остальных элементов интерфейса
3) Размещение элементов интерфейса в иерархии контейнеров
4) Обеспечение реакции элементов на события
5) Все заработало!

Если рассматривать процесс создания графических приложений в общем, то все происходит примерно так: создается основное окно, в нем размещаются компоненты графического интерфейса. Далее обеспечивается реакция компонентов на события, и можно считать, что все работает.

Класс java.awt.Component

- Компонент (widget, control) — отображаемый и взаимодействующий с пользователем элемент GUI
  - java.awt.Component - абстрактный класс — элемент GUI
  - размер, цвет, местоположение
  - порождает основные события

Хотя AWT считается устаревшей библиотекой, с помощью нее приложения давно не разрабатывают, но многие классы Swing основаны на абстракциях AWT. Одной из таких абстракций является понятие компонента.

Компонент — это элемент интерфейса, который может отображаться и взаимодействовать с пользователем. Абстрактный класс java.awt.Component определяет основные характеристики, такие как размер, цвет и местоположение, какого-то абстрактного компонента. Компонент также порождает основные события.

Цвет

- Класс Color
  - Константы
    - Color.BLACK
    - Color.RED
    - ...
  - Конструкторы
    - Color(r, g, b [,a]) int (0-255), float (0.0-1.0)
    - Color(int [,boolean]) int (0x[AA]RRGGBB)
  - Методы
    - getRed(), getGreen(), getBlue(), getAlpha()
    - brighter(), darker()

255,0,0	0.5, 0, 0	0xFFA0A0	0,0,0
255,255,0	0.5, 0.5, 0	0xFFFFA0	0.2,0.2,0.2
0,255,0	0, 0.5, 0	0xA0FFA0	0x666666
0,255,255	0, 0.5, 0.5	0xA0FFFF	153,153,153
0,0,255	0, 0, 0.5	0xA0A0FF	0.8,0.8,0.8
255,0,255	0.5, 0, 0.5	0xFFA0FF	0xFFFFFF

8

Цвет компонента характеризуется классом Color, в котором есть набор констант для основных цветов, можно создать цвет с помощью конструкторов. Цвет задается либо с помощью трех составляющих RGB, возможно с указанием прозрачности. Значения составляющих могут быть от 0 до 255, либо от 0 до 1. Можно задавать цвет в виде 16-ричного целого числа, можно задавать с помощью трех отдельных значений. Есть методы, которые позволяют получить отдельные составляющие части цвета, а есть методы, которые позволяют получить цвет светлее или темнее текущего.

Методы класса Component

- Цвет текста и цвет фона

```
Color getForeground()
void setForeground(Color)
Color getBackground()
void setBackground(Color)
```


Соответственно, у каждого компонента есть методы, которые позволяют управлять его цветом - получать или устанавливать основной или фоновый цвет.

Положение и размер

- Класс Point (int x, int y)
  - getX(), getY(),
  - setLocation(x,y)
- Класс Dimension (int height, int width)
  - getHeight(), getWidth(),
  - setSize(h, w)
- Класс Rectangle (int x, int y, int height, int width)
  - getX(), getY(), getHeight(), getWIdth(), getLocation(), getSize()
  - setLocation(x,y), setSize(h,w), setBounds(x,y,h,w)

Положение и размер компонента характеризуются прямоугольником, в который вписывается компонент. Чтобы задать прямоугольник, нужно указать его положение (координаты верхней левой точки), и размер (высоту и ширину). Положение (Location) задается классом Point - точка с координатами x, y. Размер (Size) задается классом Dimension, представляющим высоту и ширину. Еще одной характеристикой можно считать границы компонента (Bounds), которые совмещают положение и размер, и задаются с помощью класса Rectangle - прямоугольник с начальной точкой и размерами.

Методы класса Component

- Положение и размеры

```
void setBounds(Rectangle)
Rectangle getBounds()
void setLocation(Point)
Point getLocation()
void setSize(Dimension)
Dimension getSize()
```


У компонента есть методы, которые позволяют задать или получить ограничивающий прямоугольник — setBouunds и getBounds, а также отдельно получить или изменить положение начальной точки и размер компонента.

Шрифт

- Класс Font
  - физические (Arial, Times)
  - логические (Dialog, DialogInput, Serif, SansSerif, Monospaced)
  - Константы:
    - Font.DIALOG, Font.MONOSPACED, Font.SERIF, Font.SANS_SERIF
    - Font.PLAIN, **Font.BOLD**, *Font.ITALIC*
  - Конструктор Font(String name, int style, int size)
  - Методы
    - String getFontName(), int getStyle(), int getSize()

У компонента может быть задан шрифт. Шрифты задаются названием гарнитуры, либо логическим обозначением. В качестве логических выделено 5 условных шрифтов: Dialog и DialogInput используются для выдачи сообщений пользователю, и отображения ввода пользователя. Serif - шрифт с засечками, SansSerif - шрифт без засечек, Monospaced - шрифт с постоянной шириной символов, который можно использовать, к примеру, для отображения исходного кода Есть константы, которые позволяют задать шрифт. Есть конструктор, есть методы для изменения параметров шрифта.

Методы класса Component

- Шрифт

```
Font getFont()
void setFont(Font)
```


У класса Component есть методы getFont() и setFont().

Методы класса Component

- Видимость

  `boolean isVisible()`

  `void setVisible(boolean)`

  - Компоненты изначально видимы, кроме основных окон
- Активность

  `boolean isEnabled()`

  `void setEnabled(boolean)`

  - Компоненты изначально активны (воспринимают действия пользователя и порождают события)

К характеристикам компонентов можно отнести видимость и активность. По умолчанию компоненты почти все изначально видимые, кроме основных окон, к которым относятся Frame и Window. Они после создания невидимые. Соответственно, для компонента можно вызвать метод setVisible с параметром true или false. Компонент может быть активным и неактивным. Активный компонент воспринимает действия пользователя и порождает какие-то события. Неактивный компонент, соответственно, этого не делает. Активность задается методом setEnabledю Методы isVisible и IsEnabled позволяют проверить видимость и активность компонента.

Методы класса Component

- Дополнительное рисование

```
void paint(Graphics)
void update(Graphics)
void repaint()
```

  - Graphics — графический контекст компонента
- 2 варианта вызова paint
  - Системный — первое отображение, изменение размера, необходимость перерисовки
    - JVM вызывает paint(Graphics)
  - Программный — изменение состояния компонента
    - в программе вызывается repaint()
    - регистрируется событие отрисовки
    - JVM вызывает update(Graphics)

15

Методы для рисования тоже находятся в классе Component. Это paint, update и repaint. Методам paint и update передается объект класса Graphics — это графический контекст компонента, набор пикселей который определяет как этот компонент выглядит.

Метод paint может вызываться двумя способами:

1) системным, когда paint вызывается при первом отображении компонента или при необходимости перерисовки из-за перемещения окон, либо изменении размера компонента;

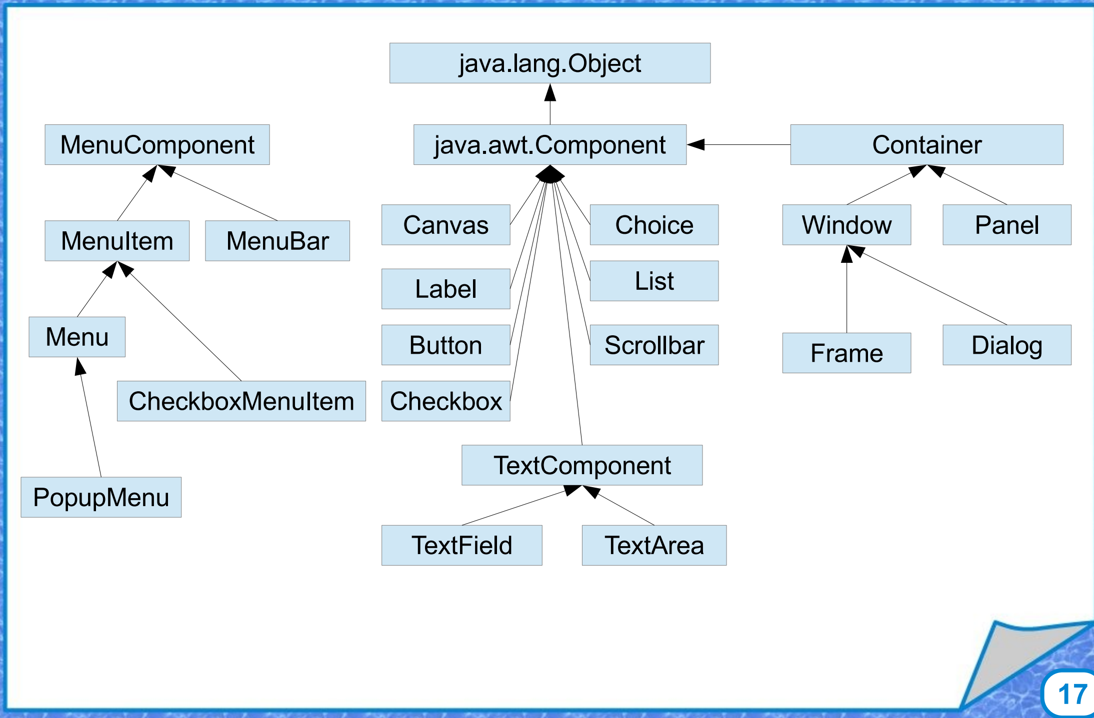
2) программным, если мы сами изменили картинку компонента, например, при анимации, тогда в программе нужно вызвать repaint(), который занесет в очередь событие для перерисовки компонента и через какое-то время вызовется paint().

Класс java.awt.Container

- Контейнер — компонент, который содержит другие компоненты
- extends java.awt.Component
- Иерархия компонентов - дерево
- Компонент может находиться только в одном контейнере
- Методы:
  - add(Component)
  - setLayout(LayoutManager)
  - validate()

16

Класс Container - это потомок класса Component, который может содержать другие компоненты. Простейший контейнер - это панель, на которой расположены компоненты. Каждый компонент может находиться только на одном контейнере. У класса Container есть методы add(), setLayout(), validate() и многие другие, но эти три являются основными, которые используются наиболее часто.



На данном слайде показана иерархия классов AWT. От класса Object наследуется Component, от которого наследуется обычные компоненты, например Canvas — пустой компонент для рисования, Label — метка, Button — кнопка, Choice — список выбора, List — выпадающий список, Scrollbar — элемент со скроллингом. Текстовые компоненты — однострочные (TextField) и многострочные (TextArea). Отдельно в иерархии стоят компоненты меню. От класса Component наследуется Container, от него наследуется Panel и Window. Панель — это универсальный контейнер без каких-то декоративных элементов. Окно дополнительно имеет элементы управления. У Window два наследника - Frame и Dialog. Диалог — это зависимое окно, которое появляется внутри других окон, а фрейм — окно верхнего уровня, которое может закрываться, сворачиваться и разворачиваться.

Размещение компонентов в контейнере

- Абсолютное позиционирование
  - Отсутствует реакция на изменение размера контейнера
  - Проблемы с изменением шрифта или локали
- Менеджер компоновки
  - Управляет взаимным расположением и размером компонентов

18

Разместить компоненты в контейнере можно двумя способами - плохим и хорошим. Плохой способ - это абсолютное позиционирование, когда, например, вы указываете, что такая-то кнопка должна располагаться на расстоянии 20 пикселей от левой границы экрана и 10 пикселей от верхней границы, размер она должна иметь такой-то и т.д. С одной стороны, вроде бы все удобно и несложно, но при этом отсутствует реакция на изменения размера контейнера. При изменении размера окна компонент все так же будет находиться в том же месте и иметь тот же размер.

Чтобы этих проблем избежать, есть классы, которые называются менеджерами компоновки. Они управляют размером компонентов и их взаимным расположением.

Менеджер компоновки

- Интерфейс LayoutManager
  - Container.setLayout(LayoutManager)
  - Container.add(Component)
- Интерфейс LayoutManager2
  - Container.setLayout(LayoutManager2, Object constraints)
  - Container.add(Component, Object constraint)

Для реализации менеджеров компоновки есть два интерфейса - LayoutManager и LayoutManager2. Отличаются они тем, что во втором случае есть объект constraints, который позволяет задать какую-то дополнительную характеристику для расположения. Соответственно есть методы setLayout и add. Метод setLayout нужен для задания контейнеру менеджера компоновки, который будет управлять расположением компонентов, а метод add() добавляет компонент в контейнер, причем расположением элемента будет управлять менеджер компоновки.

ИТМО ВТ

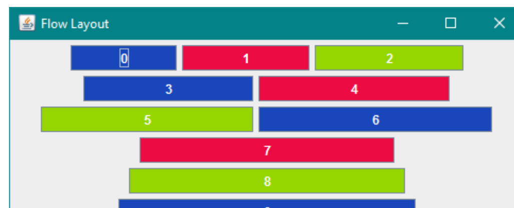
Менеджер компоновки

- Расстановка элементов
  - Container.validate()
  - Container.invalidate()
  - Container.doLayout()
  - LayoutManager.layoutContainer(Container)
- Управление размером компонентов
  - Component.getPreferredSize()
  - Component.getMinimumSize()
  - Component.getMaximumSize()

Метод validate() проходит по дереву компонентов и валидирует их размеры и расположение. Фактически расстановкой занимается метод doLayout(). Для управления размерами компонентов используются 3 характеристики - минимальный размер, максимальный размер и предпочтительный размер. Их можно установить и можно получить их значения. Менеджер компоновки использует эти значения для определения размера компонентов.

FlowLayout

- Заполнение контейнера слева направо (или справа налево) построчно
- Компоненты сохраняют свой размер preferredSize
- Управление размещением:
  - `setHgap(int), setVgap(int) // 5`
  - `setAlignment(LEFT, RIGHT, CENTER)  // CENTER`

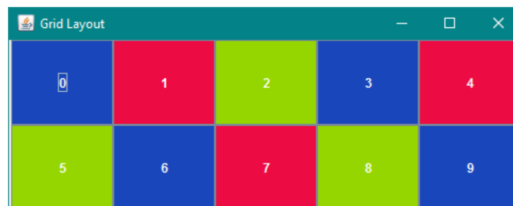


Рассмотрим стандартные менеджеры компоновки AWT. FlowLayout — самый простой менеджер компоновки. Контейнер заполняется слева направо построчно, если компонент не влезает в очередную строку, начинается следующая строка. Особенность этой компоновки в том, что компоненты сохраняют свой предпочитаемый размер. Поэтому его удобно использовать для сохранения размера единственного компонента. Можно устанавливать промежутки по горизонтали и вертикали, а также выравнивание по левому или правому краю или по центру.

ИТМО ВТ

GridLayout

- Контейнер делится одинаковые ячейки по строкам и столбцам
- Все компоненты будут одного размера
- GridLayout(int rows, int cols)
- Управление размещением:
  - `setHgap(int), setVgap(int) // 0`
  - `setRows(int), setColumns(int) // 1, 0`

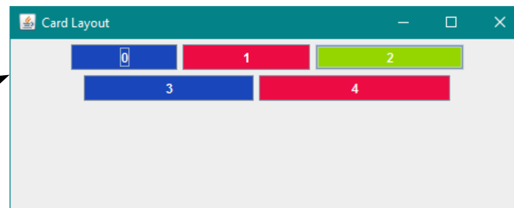


GridLayout — таблица с ячейками одинакового размера, состоящая из строк и столбцов. Точно также можно задать размер промежутков, можно установить количество строк и столбцов, Дальше добавляются элементы, заполнение идет по строкам. Если не хватает элементов, останутся пустые ячейки. Значение 0 обозначает, что строк или столбцов будет столько, чтобы поместились все элементы. Элементы при данной компоновке растягиваются под размеры ячейки.

CardLayout

- Несколько компонентов отображаются в одном месте

```
Panel p1 = new Panel(); p1.setLayout(new FlowLayout());
Panel p2 = new Panel(); p2.setLayout(new GridLayout(2,3));
CardLayout card = new CardLayout();
p.setLayout(card);
p.add(p1, "FlowPanel");
p.add(p2, "GridPanel");
card.show(p,"FlowPanel");
```

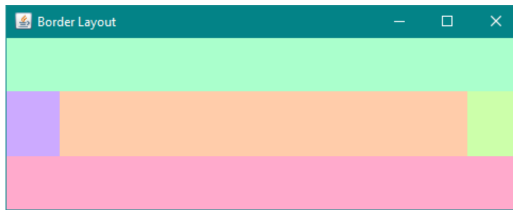


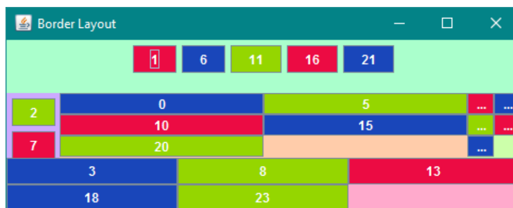

```
card.show(p,"GridPanel");
```

CardLayout — это компоновка для имитации панели с табами или вкладками. Панели добавляются в контейнер с компоновкой CardLayout, в один момент времени отображается только одна из них. Выбрать нужную панель можно с помощью метода show().

BorderLayout

- Компоненты располагаются в 5 областях:
- CENTER, NORTH, WEST, SOUTH, EAST;



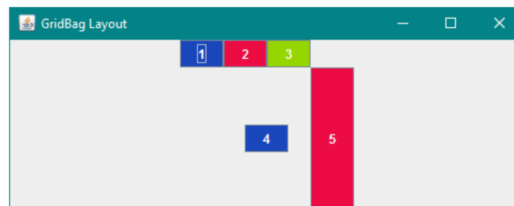


BorderLayout — часто используется для окон верхнего уровня, потому что с его помощью можно реализовать стандартную структуру окна - с центральной областью для основного содержимого, хедером, футером, и двумя боковыми панелями. Для того, чтобы поместить компонент в какую-то область, надо передать в метод add сам компонент и указать область. Если область не указать, компоненты будут попадать в центр.

GridBagLayout

- Контейнер делится на ячейки по строкам и столбцам

```
p.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.gridx = 0; c.gridy = 0; c.fill = BOTH;
p.add(new Button("1"), c);
c.gridx = RELATIVE;
p.add(new Button("2"), c); p.add(new Button("3"), c);
c.gridx = 1; c.gridy = 1; c.gridwidth = 2; c.weighty = 1.0;
c.fill = NONE;
p.add(new Button("4"), c);
c.fill = VERTICAL; c.gridx = RELATIVE;
p.add(new Button("5"), c);
```



GridBagLayout - довольно сложная компоновка, но если с ней разобраться, можно сделать почти любое расположение компонентов. основанное на сетке из ячеек. Во-первых, ячейки могут быть разной ширины, разной высоты, во-вторых, их можно объединять, можно пропускать. С этими ячейками можно делать все что угодно. В основе все равно прямоугольная сетка, но получается более гибкой.

Что еще нужно?

- Элементы созданы
- Размещены по контейнерам
- Все?
- http://knopka.ushmanov.ru/about.htm

Кнопка

Когда-то в интернете был сайт, на сайте была кнопка, прямо в центре, на нее можно было нажимать. Она ничего не делала, вообще ничего. Можно было почитать раздел "о кнопке", и там было написано, что это специальная философская кнопка, вся суть которой состоит в том, что она ничего не делает. На сайте также был форум и гостевая книга, где люди, которые нажимали на кнопку, описывали свои ощущения. У кого-то просветление наступало, у кого-то еще что-то, кнопка не увеличивала энтропию Вселенной, потому что ничего не делала, и работала без багов. Те, кто хотят повторить подобную кнопку, могут закончить изучение лекции, а кому интересно, как сделать, чтобы кнопки делали хоть что-то, могут переходить к изучению обработки событий.

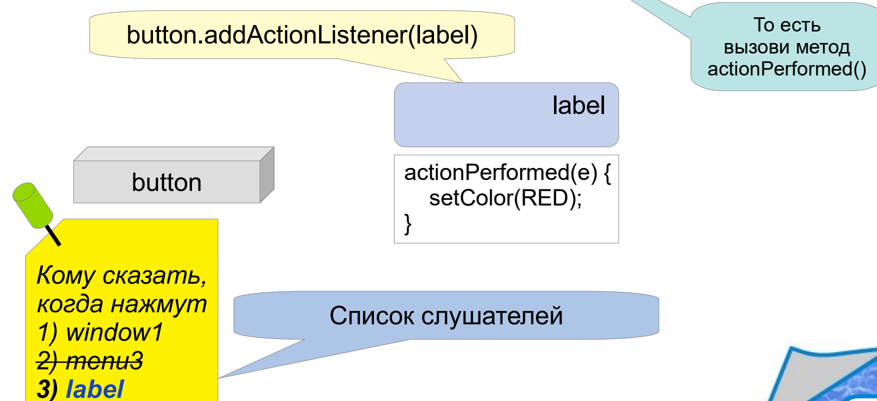
Обработка событий

- Событийно-ориентированное программирование
- Не задана последовательность выполнения кода
- Код выполняется асинхронно при наступлении определенных событий

Обработка событий обычно реализуется с помощью паттерна Observer, и такой вариант программирования называется «событийно-ориентированным», смысл его состоит в том, что последовательность исполнения кода не задается вообще, время, когда часть кода будет исполнена определяется наступланием событий. Пользователь нажимает на кнопки, выбирает элементы из списка, двигает мышкой и все эти действия вызывают асинхронное выполнение определенных участков кода.

Обработка событий

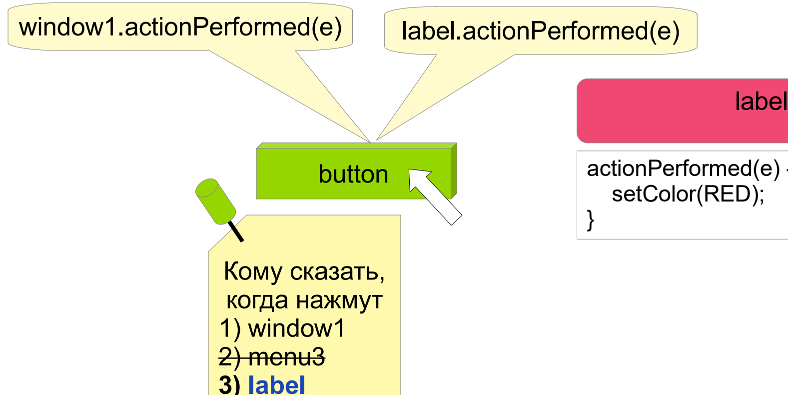
- button.addActionListener(label)
- Перевод:
  - – Кнопка, когда тебя нажмут — скажи метке, а?
  - – Ладно, записала...



Представим, что у нас есть кнопка и метка. Вызываем у кнопки метод addActionListener, которому передадим метку в качестве параметра. В переводе на человеческий язык это значит, что метка подписывается на извещения о событиях кнопки. Когда кнопку нажмут, она должна сказать об этом метке.

Обработка событий

- label.actionPerformed(new ActionEvent e)
- Перевод:
  - Меня нажали!!!
  - Окей, принято, я краснею!



Для того, чтобы это сработало, метка должна реализовывать интерфейс ActionListener, при этом у нее должен быть реализован метод actionPerformed(). Когда кнопку нажимают, кнопка радостно кричит «Меня нажали!» Этот крик передается каждому слушателю из списка с помощью вызова как раз именно этого метода actionPerformed(). И соответственно метка в этом методе реагирует на событие нажатия кнопки и перекрашивается в другой цвет.

Обработка событий

- Источник события — любой компонент
- Событие — потомок класса AWTEvent
- Обработчик события — реализует интерфейс ...Listener и соответствующие методы, в которых располагается код обработки события. Методу передается объект события

```
class A implements ActionListener {
    Button b = new Button("OK");
    Label l = new Label("Button pressed");
    l.setVisible(false);
    b.addActionListener(this);   - подписка на событие
......
public void actionPerformed(ActionEvent e) {
    l.setVisible("true");  - реакция на событие
}
```

30

Источником событий может быть любой компонент. На него могут нажать мышкой, выбрать пункт меню или сделать что-то еще. Любое событие графического интерфейса — потомок класса AWTEvent. В классе события фиксируется время, когда событие произошло, компонент, где событие произошло, и, соответствено, что именно произошло. Обработчик события реализует интерфейс Что-то-тамListener. Слушателей много и они разные. Для того, чтобы подписаться на события, нужно вызвать метод «добавить слушателя» у компонента, где ожидается событие, чтобы отреагировать на событие — компонент вызывает метод слушателя, соответствующих событий, и зависящий от интерфейса слушателя.

Обработка событий

- Анонимным классом

```
b.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    l.setVisible("true");
  }
});
```

- Лямбда-выражением

```
b.addActionListener((e) -> l.setVisible("true"));
```


Обработку событий можно реализовать анонимным классом, можно лямбда-выражением. Далее перейдем к рассмотрению конкретных слушателей.

MouseListener, MouseMotionListener, MouseEvent

- MouseListener
  - mousePressed(MouseEvent)
  - mouseReleased(MouseEvent)
  - mouseClicked(MouseEvent)
  - mouseEntered(MouseEvent)
  - mouseExited(MouseEvent)
- MouseMotionListener
  - mouseDragged(MouseEvent)
  - mouseMoved(MouseEvent)
- MouseEvent
  - getPoint()
  - getLocationOnScreen()
  - getButton()
  - getClickCount()

32

MouseListener обрабатывает статические события от мыши: нажатие кнопки, отпускание кнопки, клик, вход в облать, выход из области.

MouseMotionListener обрабатывает события движения: перемещение мыши с нажатой кнопкой и без нажатой кнопки. Событие для обоих слушателей общее - MouseEvent. При вызове методов объекта события можно узнать, в какой точке была нажата кнопка мышки, какая кнопка нажата и какой был клик - одинарный, двойной, тройной, и т. д.

Классы-адаптеры

```
• class X implements MouseListener {
     public void mousePressed(MouseEvent e) {
          // обработка нажатия кнопки мыши
      }
  // обработка других событий не требуется
       public void mouseClicked(MouseEvent e) { }
       public void mouseReleased(MouseEvent e) { }
       ....
  }
• class Y extends MouseAdapter {
      public void mousePressed(MouseEvent e) { ... }
  }
```

33

Когда используются слушатели с большим количеством методов, как MouseLIstener, а обрабатывать нужно только одно или два события, например только нажатие кнопки, удобно использовать соответствующий класс адаптер, например, MouseAdapter, в котором реализованы все методы интерфейса MouseListener, при этом они пустые и ничего не делают. При наследовании этого класса будет достаточно переопределить нужный метод и добавить в него код, чтобы не писать самостоятельно несколько пустых методов.

KeyListener, KeyEvent

- KeyListener
  - keyPressed(KeyEvent)
  - keyReleased(KeyEvent)
  - keyTyped(KeyEvent)
- KeyEvent
  - getKeyChar() // для keyTyped()
  - getKeyCode() // для keyPressed, keyReleased
  - getModifiers() // Shift, Alt, Ctrl, Meta ...
  - getKeyLocation() // Standard, Left, Right, Numpad, Unknown

34

Аналогичный слушатель есть для клавиатуры - KeyListener. Он обрабатывает следующие события - кнопка нажата, кнопка отпущена, кнопка нажата и затем отпущена (keyTyped). У события KeyEvent также можно вызвать методы для получения подробностей и деталей.

WindowListener, WindowEvent

- WindowListener
  - windowOpened(WindowEvent)
  - windowClosing(WindowEvent)
  - windowClosed(WindowEvent)
  - windowActivated(WindowEvent)
  - windowDeactivated(WindowEvent)
  - windowIconified(WindowEvent)
  - windowDeiconified(WindowEvent)
- WindowEvent
  - getNewState()
  - getOldState()
  - getOppositeWindow()

35

WindowListener - слушатель, связанный с окном. Реагирует на события - окно открыто, окно закрывается, окно закрыто, окно активировано, окно деактивировано, окно свернуто, окно развернуто. WindowEvent - соответствующий класс-событие.

ActionListener, ActionEvent

- ActionListener
  - actionPerformed(ActionEvent)
- ActionEvent
  - нажата кнопка
  - двойной клик в списке
  - выбор пункта меню
  - клавиша Enter в текстовом поле

36

Очень часто встречается ActionListener — это событие основного действия для компонентов. У многих компонентов есть основное действие: для кнопки — она нажата, для списка — двойной клик по элементу списка, для меню — выбор пункта меню, для текстового поля — нажатие клавиши Enter. В общем, какое-то действие, связанное с компонентом.

AdjustmentListener, AdjustmentEvent

- AdjustmentListener
  - adjustmentValueChanged(AdjustmentEvent)
- AdjustmentEvent
  - int getValue()
  - boolean getValueIsAdjusting()

AdjustmentListener - слушатель для компонентов, которые имеют непрерывный диапазон значений. Например, слайдер. Событие говорит о том, что значение поменялось то есть подвинули движок слайдера. При любом изменении вот этого положения слайдера, вызывается метод adjustmentValueChanged с событием AdjustmentEvent в качестве параметра.

ItemListener, ItemEvent

- ItemListener
  - itemStateChanged(ItemEvent)
- ItemEvent
  - Object getItem()
  - int getStateChange() // selected-deselected
  - установка-сброс флажка
  - установка-сброс пункта меню
  - выбор элемента списка

ItemListener — слушатель событий пунктов, например, пункта списка или пункта меню. События извещают о том, что изменилось состояние этого пункта.

TextListener, TextEvent

- TextListener
  - textValueChanged(TextEvent)
- TextEvent
  - изменился текст в текстовом компоненте

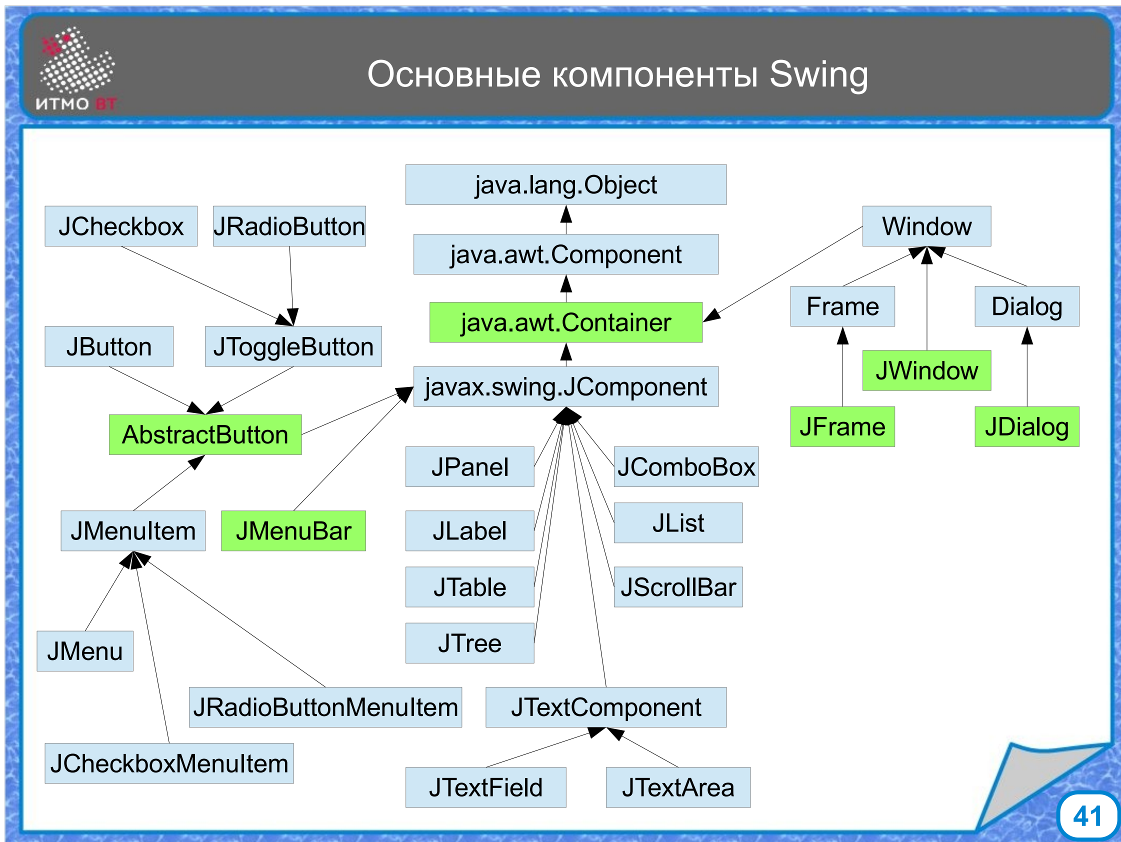
Ну и TextListener — это слушатель изменения значения текста. Был один текст, стал другой. Произошло событие TextEvent.

Основы Swing

- Функциональность такая же как в AWT, только добавлены новые классы
  - Менеджеры компоновки
  - Обработка событий
- Функциональность изменилась
  - Компоненты - легковесные, соответствующие компоненты ОС не используются.

40

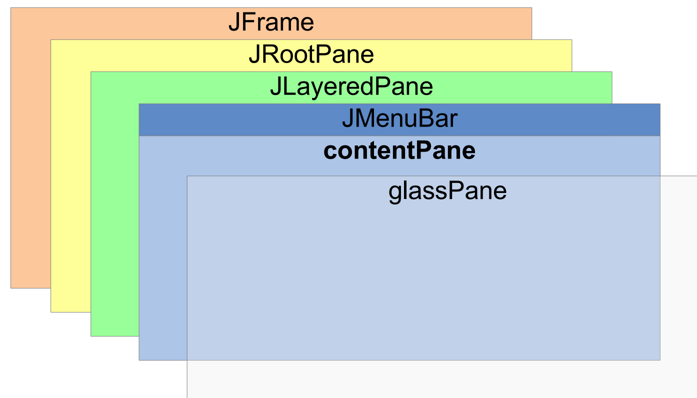
На этом можно закончить рассмотрение той части AWT, которая все еще используется. Менеджеры компоновки и события, которые были добавлены в Swing, работают точно так же. А вот компоненты отличаются. Названия классов компонентов Swing, чтобы было легче отличать их от компонентов AWT, начинаются с буквы "J". JComponent наследуется от класса java.awt.Container. Поэтому в свинговские компоненты можно легко добавлять иконки, тексты, и другие компоненты. Многие компоненты Swing аналогичны компонентам AWT по функциональности, но, если каждому компоненту AWT соответствует виджет операционной системы, который задает вид компонента и которому компонент AWT делегирует поведение, то большинство компонентов Swing - легковесные, их код отрисовки и реализации поведения написан на Java.



Основные отличия иерархии компонентов на слайде выделены. Кнопка в AWT была простой конкретной кнопкой. В Swing класс AbstractButton реализует кнопочное поведение, и от него наследуются разные виды кнопок - обычная JButton, переключающая JToggleButton, от которой в свою очередь наследуются JRadioButton и JCheckbox. По другому выглядит иерархия классов для меню. И есть три компонента Swing, которые все-таки связаны с операционной системой - это JWindow, JDialog, JFrame. Они наследуются от соответствующих контейнеров AWT Окнами управляет операционная система, поэтому они сохраняют связь с виджетами ОС.

JFrame

- Не является легковесным компонентом — это окно ОС
- Содержит набор панелей для размещения компонентов
- При создании — невидимый
- JFrame.add() = JFrame.getContentPane.add()



JFrame - основное окно свинга. Фрейм содержит набор панелей, которые располагаются одна за другой, на каждой из них может что-то отображаться. Чаще всего используется Content Pane, где располагается основной контент окна. JLayeredPane - многослойная панель, где элементы могут располагаться в разных слоях. Glass Pane — слой для размещения элементов переднего плана - тултипов, всплывающих меню, окон. Если методом add() добавлять компоненты на фрейм, они на самом деле добавляются на ContentPane.

JFrame

```
JFrame f = new JFrame();
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.add(new JLabel("Hello!"), BorderLayout.CENTER);
f.setJMenuBar(new JMenuBar());
f.pack(); // установка размеров фрейма
f.setVisible(true);
```


По традиции можно написать приложение "Hello World" с использованием Swing. Создаем JFrame, устанавливаем действие выхода при закрытии окна. Потом добавляем метку со строкой "Hello!", устанавливаем пустую полоску меню. Метод pack() рассчитывает размеры фрейма, исходя из размеров компонентов, которые на данный момент в нем находятся, чтобы все они поместились на фрейм. И, так как фрейм изначально невидимый, нужно установить ему видимость - setVisible(true).

Потоки в приложениях Swing

- Основные потоки (initial)
  - Для выполнения основного кода приложения (main)
  - Запуск метода для создания элементов интерфейса в потоке обработки событий
- Поток обработки событий (event-dispatching)
  - для кода обработки событий
  - Все действия с элементами интерфейса здесь!
  - Действия должны выполняться быстро
- Фоновые потоки (worker)
  - для задач, требующих длительного времени
  - запускаются с помощью класса SwingWorker

44

Swing и другие графические библиотеки являются многопоточными. Пользователи не хотят ждать, когда приложение будет готово обработать клик мышкой. Для того, чтобы приложение корректно работало, действия должны выполняться в соответствующих потоках. Все события обрабатываются в специальном потоке, который называется event-dispatching thread, поток обработки событий. Почти все компоненты интерфейса не потокобезопасны, поэтому все действия с ними нужно производить в этом отдельном потоке. Обычно обработка события происходит быстро, и задержек не происходит. Если нужно выполнить какое-то долгое действие, например, загрузку файла, для него желательно выделить отдельный фоновый поток, который управляется классом SwingWorker.

ИТМО ВТ

Организация приложения Swing

```
public class Main {
  public static void main(String... args) {
    SwingUtilities.invokeLater(() -> gui());
  }
  private void gui() {
    JFrame f = new JFrame();
    ...
    f.setVisible(true);
  }
}
```

- InvokeLater(Runnable) — асинхронный запуск задачи (для приложений)
- InvokeAndWait(Runnable) — синхронный запуск задачи (для апплетов)

45

Для правильной органиации приложения Swing компоненты интерфейса должны создаваться в потоке обработки событий. Для этого проще всего написать отдельный метод (в примере gui()), в нем разместить код создания компонентов, и в методе main() передать объект типа Runnable с методом run(), содержащим вызов метода gui(), статическому методу invokeLater() класса SwingUtilities. Это действие поместит исполнение метода gui() в очередь потока обработки событий, где он исполнится позже, когда до него дойдет очередь. Все эти потоки управляются Swing-ом, заботиться о них не надо.

В методе gui() можно разместить код, создающий и отображающий метку "Hello!".

SwingWorker<T, V>

- T — тип результата
- V — тип промежуточных значений
- abstract T doInBackground() - метод для выполнения задачи в фоновом потоке
- done() - метод, вызываемый после выполнения
- T get() - возвращает результат
- publish(V) — передать промежуточный результат
- process(List<V>) - обработать
- Свойства
  - state (PENDING, STARTED, DONE)
  - progress (0 - 100)
- addPropertyChangeListener()

46

Что делать с длинными задачами? Для них можно использовать класс SwingWorker. Основной его метод - doInBackground(), запускающий задачу в фоновом потоке. Метод done() вызывается после завершения выполнения задачи. Метод get() ждет завершения задачи и возвращает результат (SwingWorker реализует интерфейс Future). Методы publish() и process() позволяют работать с промежуточными результатами.

Свойства класса - state, которое показывает состояние задачи, и progress (от 0 до 100) - на сколько процентов выполнена задача. Можно подписать слушателей на событие изменение свойств с помощью вызова метода addPropertyChangeListetner().

JComponent

- extends java.awt.Container — может содержать картинку
- всплывающие подсказки — setToolTipText()
- построение основано на шаблоне MVC
- встроенная двойная буферизация при отрисовке

Класс JComponent расширяет java.awt.Container. У него расширенная функциональность по сравнению с компонентами AWT. Можно задать всплывающие подсказки, структура компонента Swing основана на шаблоне MVC, который будет рассмотрен позже, компонент имеет встроенную двойную буферизацию при отрисовке содержимого.

paintComponent

- java.awt.Component

```
paint(Graphics g) { // код для рисования }
```

- javax.swing.JComponent

```
paint(Graphics g) {
  paintComponent(g);
  paintBorder(g);
  paintChildren(g);
}
paintComponent(Graphics g) {
  ui.update(g, this)
}
```

- javax.swing.plaf.ComponentUI

```
update(Graphics g, Component c) {
 // заполняет фон цветом фона
 this.paint(g, c);  // отрисовка компонента
}
```

48

Рисование происходит следующим образом. Для AWT в классе Component есть метод paint(), который нужно переопределить. Объект класса Graphics передается в paint и используется для вызова примитивов рисования. В Swing в методе paint() происходит вызов 3 методов: paintComponent, paintBorder и paintChildren. Переопределять нужно paintComponent(). Его стандартная реализация вызывает метод ui.update(), который корректно отображает вид компонента. Обычно свой компонент для рисования наследуется от какого-то стандартного, поэтому для сохранения отрисовки можно воспользоваться родительским методом.

Свой код для рисования

- paintComponent(Graphics2D g) {

```
super.paintComponent(g);
Graphics2D g2d = (Graphics2D) g;
g.drawLine(...);
```

}

для вызова метода отрисовки

- repaint()

В методе paintComponent() потомка нужно для начала вызвать этот же метод у суперкласса. Тогда он отрисует сам компонент, дальше можно развлекаться рисованием. Метод paintComponent получает объект класса Graphics. На самом деле обычно передается объект класса Graphics2D с расширеной функциональностью. У этого объекта есть большое количество методов для рисования примитивов. Для того, чтобы компонент обновился, нужно вызвать метод repaint(). Он добавит вызов метода paint() в очередь на обновление картинки.

ИТМО ВТ

Анимация

- Метод отрисовки объекта

  paintComponent(Graphics g) { drawObject(g); }
- Метод изменения объекта (размера, координаты, цвет)

  change() { x++; y--; color.darker(); width += 2; }
- Аниматор
  - javax.swing.Timer
    - + actionPerformed() { change(); repaint(); }
  - java.util.Timer
    - + TimerTask.run() { change(); repaint(); }
  - Thread
    - + Runnable.run() { change(); repaint(); sleep(); }

50

Для того, чтобы реализовать анимацию надо:

1) написать метод отрисовки, например drawObject, который мы будем вызывать в методе paintComponent.

2) написать метод изменения состояния объекта, например, change, где мы будем на каждом шаге анимации менять координаты, цвет и размер.

3) призвать аниматора, тут есть много вариантов. Например, можно использовать java.swing.Timer который умеет генерировать события ActionEvent через определенные промежутки времени, либо использовать стандартный таймер java.util.Timer. Ну и метод для суровых студентов ИТМО - пишем свой аниматор с помощью класса Thread, задавая паузу методом sleep().

MVC и Swing

- MVC — Model, View, Controller
  - Модель отвечает за поведение
  - Представление — отвечает за отображение
  - Контроллер — связывает модель и представление и управляет ими
- Реализация Swing — Model + UI Delegate
  - UI Delegate = View + Controller
  - Модель может быть визуальной или моделью данных
  - Одну модель данных можно назначить разным компонентам
  - В случае большого числа событий можно использовать ChangeEvent — изменение в модели.

Шаблон MVC (Model-View-Controller) - это шаблон, определяющий структуру. При его использовании выделяют 3 элемента - модель, представление и контроллер. Модель отвечает за логику поведения, предсталвение отвечает за внешний вид, а контроллер связывает модель с представлением и отвечает за реакцию на события. Основная идея - разделить модель и представление. В Swing применяется чуть урезанная схема, представление и контроллер объединяются в так называемый UIDelegate. Модель может быть либо визуальной, либо моделью данных. Визуальная модель представляет видимые свойства (кнопка нажата или нет). Модель данных представляет структуру данных, лежащую в основе компонента. Например, массив строк для представления элементов списка. Одна и та же модель может назначаться нескольким компонентам.

Модели

- Модели — интерфейсы: ButtonModel, ListModel, ...
- Реализации моделей по умолчанию — классы, например: DefaultListModel, DefaultTableModel
- Для сложных моделей дополнительно имеются классы абстрактных моделей, предназначенные для облегчения написания своих классов на их основе. Например, AbstractTableModel, AbstractTreeModel

Модели реализованы в виде интерфейсов, на каждый вид компонента есть своя модель, например для списка - ListModel. Для каждого интерфейса модели реализована модель по умолчанию, которую обычно используют, например, для списка - DefaultListModel. Для сложных моделей дополнительно есть еще абстрактные модели. Дефолтная модель - модель со стандартным поведением для большинства случаев, Абстрактная модель - шаблон для реализации своей собственной модели.

Делегаты

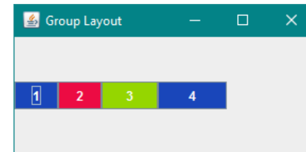
- Делегаты — потомки класса javax.swing.plaf.ComponentUI, например, ButtonUI, ListUI
- Напрямую в коде не используются
- Для управления делегатами предназначен класс javax.swing.UIManager

```
UIManager.setLookAndFeel(
   UIManager.getSystemLookAndFeelClassName());
SwingUtilities.updateComponentTreeUI(frame);
frame.pack();
```


Делегаты напрямую не используются, обычно ими управляет класс UIManager, который в основном использует делегаты для того, чтобы поменять внешний вид компонентов. На слайде показано, как это можно сделать.

Менеджеры компоновки Swing

- BoxLayout
  - Компоненты располагаются в один ряд вертикально или горизонтально
- Класс Box — контейнер с BoxLayout
  - Box.createHorizontalBox()
  - Box.createVerticalBox()
- createRigidArea(Dimension)
- createHorizontalGlue()
- createVerticalGlue()
- Filler(minSize, prefSize, maxSize) - заполнитель



54

Менеджеры компоновки Swing — это добавочные менеджеры, которых не было в AWT.

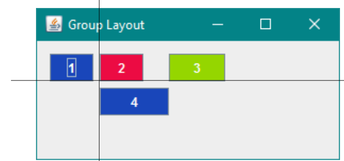
BoxLayout — это компоновка в один ряд, либо вертикально, либо горизонтально. К нему прилагается специальный контейнер Box с заданной компоновкой. Соответственно, можно создать горизонтальный или вертикальный бокс.

Методом createRigidArea можно задать фиксированное расстояние между компонентами, а методами createHorizontalGlue и createVerticalGlue задать гибкое расстояние, и с помощью класса Filler установить заполнитель.

Менеджеры компоновки Swing

- GroupLayout
  - Все компоненты описываются дважды — горизонтальное расположение и вертикальное расположение
  - Все компоненты являются участниками групп — последовательных и параллельных

```
GroupLayout gl = new GroupLayout(p);
gl.setVerticalGroup(
   gl.createSequentialGroup()
   .addGroup(gl.createParallelGroup(GroupLayout.Alignment.BASELINE)
               .addComponent(c1)
               .addComponent(c2)
               .addComponent(c3))
   .addComponent(c4)
);
gl.setHorizontalGroup(
   gl.createSequentialGroup()
     .addComponent(c1)
     .addGroup(gl.createParallelGroup(GroupLayout.Alignment.LEADING)
                 .addCompoment(c2)
                 .addComponent(c4))
     .addComponent(c3)
);
```

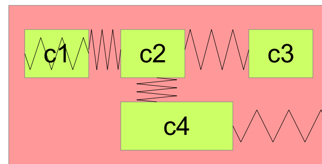


GroupLayout — это более сложный менеджер компоновки. Здесь расположение всех компонентов задается по два раза. Горизонтальное расположение и вертикальное расположение. Компоненты объединяются в последовательные и параллельные группы. Например, если рассматривать горизонтальное расположение, то элемент 1 находится в одной последовательной группе с элементами 2 и 3. И эта группа параллельна группе с элементом 4. А если рассматривать вертикальное расположение, то есть три параллельные группы, в первой - элемент 1, во второй - последовательная группа элементов 2 и 4, и в третьей - элемент 3. В коде задаются группы, а потом менеджер компоновки, исходя из этих условий, располагает компоненты в контейнере.

ИТМО ВТ

Менеджеры компоновки Swing

- SpringLayout
  - Все компоненты соединены пружинами (Spring), которые имеют минимальную, максимальную и предпочтительную длину
  - Между краями соседних компонентов устанавливаются соответствующие пружины, в итоге получается компоновка, в определенных пределах растягивающаяся и сжимающаяся
  - Обычно используется автоматическими расстановщиками



И SpringLayout — это компоновка, в которой условно считается, что компоненты соединяются невидимыми пружинами, у которых заданы 3 значения: минимальная длина, максимальная длина, предпочтительная длина. Пружины соединяют либо 2 компонента, либо компонент и границу контейнера. Компоновщик расставляет элементы в зависимости от длины пружин. При изменении размеров контейнера пружины могут растягиваться и сжиматься в пределах заданных ограничений длины.

JLabel

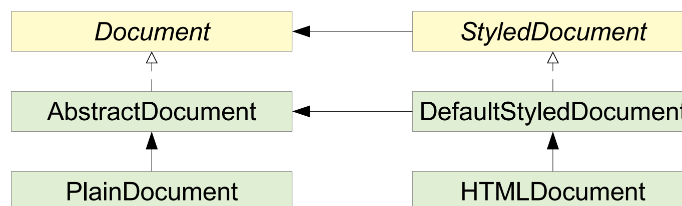
- Метка изначально прозрачная
- метод setOpaque(true) — сделать непрозрачной

Кратко рассмотрим некоторые особенности компонентов. Метка JLabel изначально прозрачная, ее можно сделать непрозрачной методом setOpaque(true).

ИТМО ВТ

Простые текстовые элементы

- JTextField — однострочное поле
  - JFormattedTextField — возможность проверки ввода
  - JPasswordField — не отображает введенные символы
  - Основное событие — ActionEvent
- JTextArea — многострочное поле
  - События — ActionEvent, UndoableEditEvent
- Модель для всех текстовых компонентов — Document



JTextField — однострочное текстовое поле. JTextArea — многострочная текстовая область. Для всех текстовых компонентов в качестве модели используется Document.

Расширенные редакторы

- EditorPane — панель текста определенного формата с заданным редактором (поддерживается редактирование обычного текста, RTF, HTML)
  - TextPane — панель текста с поддержкой стилей
  - HyperlinkListener

В Swing имеется класс, представляющий панель с заданным редактором, в том числе текста в формате HTML.

JButton, JCheckBox, JRadioButton

- Конструктор принимает строку
- Для JCheckBox и JRadioButton — еще состояние (boolean)
- JRadioButton используется в группе ButtonGroup
- События —
  - ActionEvent для JButton, JRadioButton
  - ItemEvent для JCheckBox (позволяет отследить select-deselect)
- Модель — DefaultButtonModel — элемент с двумя состояниями
- Почти так же обрабатываются JMenuItem, JMenu, JCheckBoxMenuItem, JRadioButtonMenuItem

60

Для кнопок в конструкторе указывается надпись на кнопке. Для JCheckbox и JRadioButton дополнительно задается начальное состояние. Радиокнопки объединяются в группы, в пределах группы активной может быть только одна радиокнопка. Модель DefaultButtonModel имеет два состояния: нажато и не нажато. С этой же моделью могут работать некоторые виды пунктов меню.

ИТМО ВТ

JList

- Конструктор принимает массив или вектор объектов
- Основное событие — ListSelectionEvent
- Модели
  - ListModel ← AbstractListModel ← DefaultListModel
    - getElementAt()
    - getSize()
  - DefaultListModel — модель данных (вектор),
  - ListSelectionModel ← DefaultListSelectionModel
  - DefaultListSelectionModel — модель вариантов выбора (одиночный, интервальный, множественный)

61

Список JList имеет 2 модели - модели данных списка (условно это массив или вектор), определяющая элементы списка, и модель выбора - ListSelectionModel, которая определяет вариант работы выбора - одиночный, множественный, групповой, интервальный.

JComboBox

- Может быть редактируемым и нередактируемым
- Конструктор принимает массив или вектор объектов
- Основное событие — ActionEvent, иногда ItemEvent
- Модель DefaultComboBoxModel реализует 3 интерфейса — ListModel, ComboBoxModel и MutableComboBoxModel.
- По сравнению с ListModel — ComboBoxModel вводит понятие выбранный элемент (отображаемый)
- MutableComboBoxModel позволяет добавлять и удалять элементы

Выпадающий список - JComboBox. В конструктор при его создании передается массив или вектор. Основное событие комбобокса - ActionEvent, срабатывающее при выборе элемента. Модель по умолчанию реализует 3 интерфейса - ListModel, определяющую данные, ComboBoxModel, который дает возможноть отобразить один выбранный элемент, и MutableComboBoxModel, который позволяет добавлять и удалять пункты.

JSpinner

- Составной компонент — 2 кнопки и редактор значений
- Конструктор принимает модель SpinnerModel
- Основное событие — ChangeEvent
- 3 готовых модели — SpinnerListModel, SpinnerDateModel, SpinnerNumberModel + AbstractSpinnerModel
- 3 готовых редактора — JSpinner.ListEditor, JSpinner.DateEditor, JSpinner.NumberEditor

63

Спиннер — это компонент со значением какого-то типа, у которого есть возможность увеличивать или уменьшать значение с помощью мелких кнопок со стрелками вверх-вниз. У спиннера есть 3 вида моделей - для чисел, дат и строк. Основным событием, которое генерирует спиннер, является ChangeEvent.

JSlider

- Движок
- Конструктор принимает min и max значения
- Основное событие — ChangeEvent
- Модель — DefaultBoundedRangeModel — еще используется для JProgressBar

Слайдер — это движок, который можно перемещать от минимального значения до максимального. Событие - тоже ChangeEvent, а модель представляет ограниченный интервал, называется DefaultBoundedRangeModel и используется также для компонента JProgressBar.

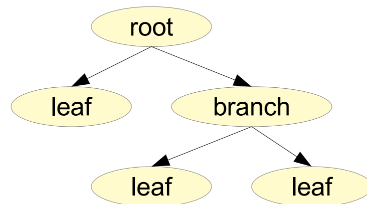
JTable

- TableModel ← AbstractTableModel ← DefaultTableModel
- DefaultTableModel — простая таблица
  - DefaultTableModel(Object[][] data, Object[] colNames)
  - DefaultTableModel(Vector data, Vector colNames)
- AbstractTableModel
  - Реализовать методы:
    - int getRowCount(), int getColumnCount(), Object getValueAt(int, int)
    - Class getColumnClass(), isCellEditable(r,c), setValueAt(r,c)
- TableModelEvent
- Сортировка — TableRowSorter

Таблица - более сложный компонент, у нее в дополнение к дефолтной модели есть абстрактная модель, которая реализует методы по отслеживанию изменений данных в таблице, предоставляя возможность переопределить методы по управлению данными, задать возможность валидации данных, сортировки и разных способов редактирвания.

JTree

- TreeModel ← DefaultTreeModel
- TreeNode ← MutableTreeNode ← DefaultMutableTreeNode
- TreePath — путь к узлу
- TreeModelEvent, TreeSelectionEvent, TreeExpansionEvent



Компонент JTree представляет собой раскрывающееся дерево узлов, опять же с разными вариантами использования моделей, в том числе модели данных, модели выбора и модели расширения дерева.

Контейнеры

- JPanel - универсальный контейнер — FlowLayout
- Box - BoxLayout
- JScrollPane — контейнер со скроллерами
- JSplitPane — контейнер из 2 частей
- JTabbedPane — контейнер с табуляторами
  - SingleSelectionModel

Рассмотрим контейнеры Swing. Простая панель JPanel — это универсальный контейнер, у которого по умолчанию установлен FlowLayout. Box — это контейнер с BoxLayout. JScrollPane — это контейнер, у которого добавлены скроллбары, JSplitPane — это контейнер, разделенный на 2 части либо по горизонтали, либо по вертикали. Их можно обменивать. И, наконец, JTabbedPane — контейнер со встроенными вкладками.

JavaFX

- JavaFX — новая библиотека для разработки RIA (Rich Internet Applications)
- Поддержка XML для создания интерфейса
- Поддержка стилей CSS
- Поддержка 2D- и 3D-графики
- Легковесные компоненты
- Интеграция с библиотекой Swing

JavaFX (которая превратилась в OpenJFX – это новая библиотека для графики. Поддерживает XML, поддерживает стиль CSS, поддерживает 2D и 3D графику. Также FX может работать с компонентами Swing. Интеграция работает в обе стороны, то есть и в Swing можно использовать компоненты JavaFX, и в JavaFX можно использовать компоненты Swing.

javafx.application

- javafx.application.Application — класс-предок всех приложений JavaFX
  - void init() — инициализация приложения (стартовый поток)
  - abstract void start(Stage s) — основной поток приложения
  - void stop() — освобождение ресурсов
  - public static void launch(String[] args) — запуск приложения

69

Основной класс для приложений JavaFX - Application. Это класс-предок всех приложений. Для написания своего приложения просто наследуемся от Applicaiton. У этого класса есть четыре основных метода:

1. init() - для инициализации, обычно туда помещается код, который задает начальные значения.
2. stop() освобождает ресурсы, вызывается при закрытии приложения.
3. абстрактный метод start(). В нем должен быть весь код приложения. Методу старт передается объект класса Stage, создавать этот объект не надо.
4. launch() запускает приложение. Этому методу можно передать аргументы

javafx.stage

- javafx.stage.Stage — основная платформа
- Контейнер верхнего уровня (аналог JFrame)
- Предоставляется системой при запуске приложения
- Обеспечивает связь с графической подсистемой ОС
  - setTitle(String)
  - setScene(Scene)
  - show()

Класс Stage - подмостки или основная платформа для различных сцен. Stage - это некий аналог основного окна, типа JFrame в Swing. Он обеспечивает связь с графической подсистемой ОС.

javafx.scene

- javafc.scene.Scene — контейнер для элементов сцены
- Должен быть хотя бы один объект класса Scene
- Элементы сцены — узлы (Node)
- Узлы образуют граф (scene graph)
- Граф включает не только контейнеры и компоненты, но также графические примитивы (текст и графические примитивы)
- Узел с дочерними узлами — Parent (extends Node)
- Корневой узел (root node) — узел без родительского узла

- Scene sc = new Scene(*root node*,300,150);

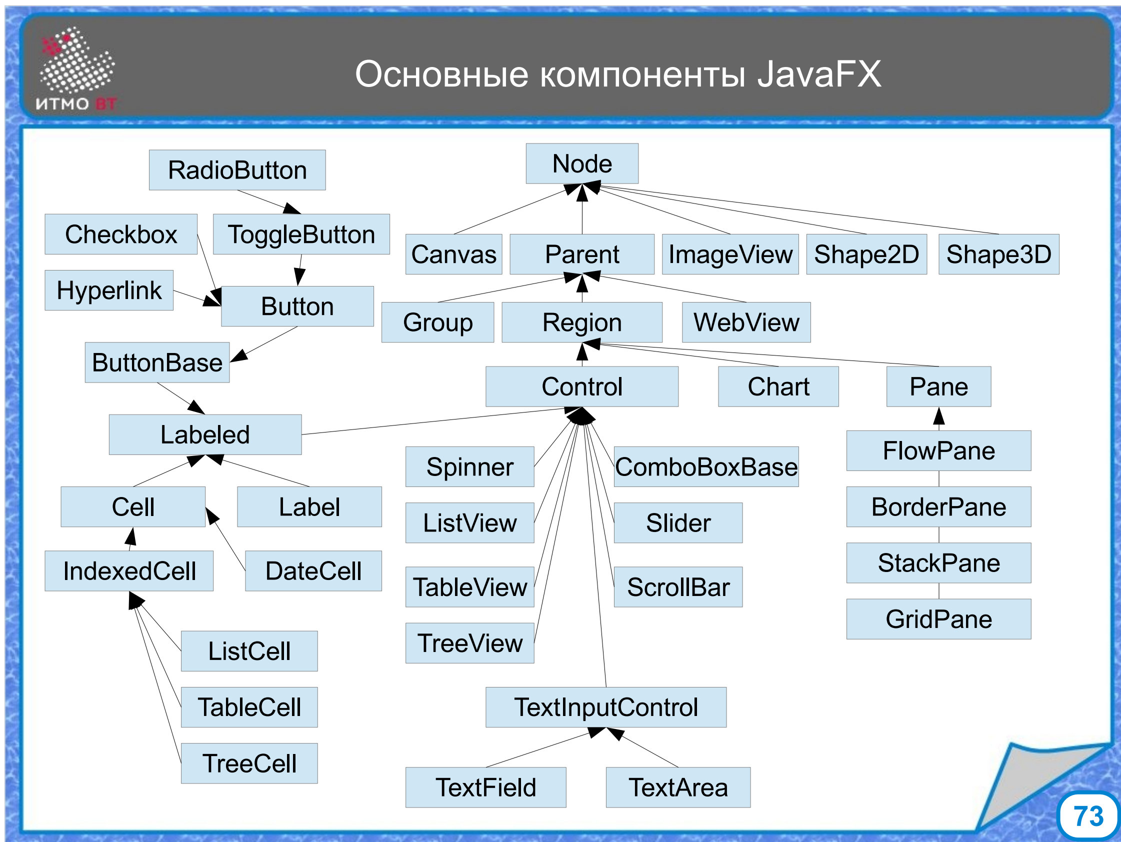
71

Класс Scene - это основной контейнер для всех элементов, Хотя бы одна сцена в приложении должна быть, иначе показывать будет нечего. Элементами сцены являются узлы (класс Node). Узлы образуют граф, Если у узла есть дочерние узлы, его представляет класс Parent. Корневой узел - узел без родителей Корневой узел обычно задается при создании новой сцены, он становится главным узлом сцены.

Node

- Свойства (properties)
  - String id
  - Parent (только один)
  - Scene
  - Стиль (styleClass, style)
  - Видимость, активность, прозрачность
  - Размеры (min, max, preferred)
  - Границы (boundsInLocal, boundsInParent, layoutBounds)
  - Трансформации (сдвиг, вращение, масштаб, наклон)
  - Эффекты
  - События (mouse, key, drag, touch, rotate, scroll, swipe, zoom)

Класс Node - узел. Узел имеет идентификатор, и один и только один родитель. У узла есть сцена, на которой он находится в данный момент.
Также у узла есть стиль, видимость, активность, прозрачность, есть размеры, границы, Узел можно подвергнуть трансформации, на него можно наложить эффекты, и задать события, которые он должен обрабатывать.



Основные компоненты JavaFX. Node и Parent уже рассмотрели. Canvas - узел для рисования, ImageView - узел с картинкой, Shape2D и Shape3D - графические примитивы. Group - наследник класса Parent, представляет из себя группу элементов, просто группу. Region — это уже что-то вроде контейнера. WebView — узел, который предназначен для отображения компонентов в браузере. Control - наследник класса Region - это любой элемент, способный взаимодействовать с пользователем. Также от класса Region наследуются Chart (диаграмма) и Pane — панель с компоновкой. Control может быть с меткой или без метки. Помеченный Control содержит текст: это кнопки, ячейки в таблицах, элементы списка, узлы дерева, ячейки таблицы. Непомеченный Control текста не содержит.

Hello World!

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;

public class Hello extends Application {
    public void start(Stage stage) {
        FlowPane fp = new FlowPane();
        fp.getChildren().add(new Label("Hello World!"));
        stage.setScene(new Scene(fp,100,200));
        stage.show();
    }
    public static void main(String... args) {
        launch(args);
    }
}
```


Минимальное приложение на JavaFX.
Создаем класс, наследующийся от Application, пишем метод start(), создаем контейнер FlowPane, потом получаем список его дочерних узлов, пустой пока что, добавляем в него метку с текстом. Объекту stage, полученному от графической подсистемы задаем новую сцену, которой передаем панель, сделав ее корневым узлом, и вызываем метод show для отображения. А в методе main вызываем запуск приложения - launch().

Контейнеры с компоновкой

- BorderPane — top, bottom, left, right, center
- HBox, VBox — в один ряд по горизонтали/вертикали
- StackPane — один над другим
- GridPane — сетка (таблица)
- FlowPane — последовательно с переносом
- TilePane — равномерные ячейки
- AnchorPane — привязка к границам родителя

Контейнеры JavaFX с разными компоновками, BorderPane — это контейнер с компоновкой типа BorderLayout, HBox и VBox — однорядные контейнеры: горизонтальный и вертикальный. StackPane — это аналог CardLayout, т. е. панели расположены друг за другом. Сетки GridPane и TilePane отличаются ячейками. У GridPane они разного размера, у TilePane - одинакоые.. FlowPane - панель с FlowLayout. AnchorPane привязывает элемент к верху, к низу, к боку или к центру.

Обработка событий

- Событие: javafx.event.Event
  - ActionEvent extends Event
- Обработчик: javafx.event.EventHandler<T extends Event>
  - void handle(T event)
- Регистрация:
  - setOnAction(EventHandler<T>)

```
Label label = new Label();
Button button = new Button("Нажми меня");
button.setOnAction((ae) -> { label.setText("Спасибо");} )
```


Обработка событий работает почти также, как в Swing, тоже есть класс события, есть его обработчик, который называется EventHandler. У него есть метод handle(). Подписка на события происводится вызовом метода setOnAction, которому передается обработчик. В отличие от Swing, у всех обработчиков один метод handle().
В приведенном примере при нажатии на кнопку на метке отображается надпись "Спасибо".

FXML

```
FlowPane fp = new FlowPane();
fp.getChildren().add(new Label("Hello World!"));
stage.setScene(new Scene(fp,100,200));

FXMLLoader loader = new FXMLLoader();
loader.setLocation("file.xml");
FlowPane fp = loader.<FlowPane>load();
stage.setScene(new Scene(fp,100,200));

<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.FlowPane?>
<?import javafx.scene.control.Label?>
<FlowPane>
   <children>
      <Label text="Hello World!"/>
   </children>
</FlowPane>
```


Еще одной особенностью JavaFX является возможность задавать элементы интерфейса с помощью XML. XML-файлы загружаются с помощью класса FXMLLoader. Приведен пример, как задать метку с помощью XML.

CSS

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.FlowPane?>
<?import javafx.scene.control.Label?>
<FlowPane>
   <children>
      <Label text="Hello World!">
         <style>
           -fx-padding: 10px;
           -fx-background: rgb(255,127,255);
         </style>
   </children>
</FlowPane>
```

```
label.setStyle("-fx-background-color: #ff77ff");
label.setStyle("-fx-padding: 10px");
```


Также элементам можно задавать стили с помощью CSS.

Приложение Swing

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SwingApp {
    public SwingApp() {
        JFrame frame = new JFrame("Hello");
        JLabel label = new JLabel("");
        JButton button = new JButton("OK");
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        button.addActionListener((ae) ->
{label.setText("Привет!");});
        frame.setSize(240, 120);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> { new SwingApp(); });
    }
}
```


И в заключении 3 небольших примера графических приложений с одинаковыми функциями, релизованные соответственно: первое с помощью Swing

Приложение JavaFX

```
import javafx.application.*;
import javafx.event.*;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.effect.*;

public class FXApp extends Application {
    public void start(Stage stage) {
        stage.setTitle("Hello");
        FlowPane root = new FlowPane();
        Label label = new Label(;
        Button button = new Button("OK");
        root.getChildren().add(label);
        root.getChildren().add(button);
        button.setOnAction((ae) -> label.setText("Привет!"));
        stage.setScene(new Scene(root,240,120));
        stage.show();
    }
    public static void main(String... args) {
        launch(args);
    }
}
```


Второе - с помощью JavaFX

Приложение SWT

```
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class SWTApp {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("Hello");
        shell.setSize(240,120);
        shell.setLayout(new FillLayout(SWT.HORIZONTAL));
        Label label = new Label(shell,SWT.BORDER);
        Button button = new Button(shell, SWT.PUSH);
        button.setText("OK");
        button.pack(); label.pack();
        button.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                label.setText("Привет!");
            }
        });
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }
}
```


И третье для сравнения - с помощью SWT.